Hoja de ejercicios del Tema 1

1. Define un tipo `Vector` que permita representar secuencias de N enteros e implementa, además de subprogramas para leer y mostrar un vector:

- Un subprograma que, dado un vector, mueva sus componentes un lugar a la derecha. El último componente se moverá al 1er lugar.
- Un subprograma que, dado un vector, calcule y devuelva la suma de los elementos que se encuentran en las posiciones pares del vector.
- Un subprograma que, dado un vector, encuentre y devuelva la componente de mayor valor.
- Un subprograma que, dados dos vectores, devuelva un valor que indique si son iguales.
- Un subprograma que obtenga si alguno de los valores almacenados en el vector es igual a la suma del resto de los valores del mismo.
- Un subprograma que obtenga si alguno de los valores almacenados en el vector está repetido.

2. Dado un array de N caracteres v1, en el que no hay elementos repetidos, y otro array de M caracteres v2, donde N ≤ M, se quiere comprobar si todos los elementos del array v1 están también en el array v2. Por ejemplo, si:

v1 =	'a'	'h'	'i'	'm'		
v2 =	'h'	'a'	'x'	'x'	'm'	'i'

El resultado sería cierto, ya que todos los elementos de v1 están en v2.

3. Implementa un programa que permita realizar operaciones sobre matrices de N×N. El programa debe permitir al usuario la selección de alguna de las siguientes operaciones:

- Sumar 2 matrices.
- Restar 2 matrices.
- Multiplicar 2 matrices.
- Trasponer una matriz.
- Mostrar una matriz señalando cuáles son los puntos de silla (los *puntos de silla* de una matriz son aquellos elementos de la misma que cumplen ser el mínimo de su fila y el máximo de su columna).

Habrá también dos subprogramas para leer del teclado o mostrar en la pantalla una matriz.

4. Escribe un programa que lea dos cadenas del teclado y determine si una es un anagrama de la otra, es decir, si una cadena es una permutación de los caracteres de la otra. Por ejemplo, "acre" es un anagrama de "cera" y de "arce". Ten en cuenta que puede haber letras repetidas en la cadena ("carro", "llave").

5. Escribe un programa que lea del teclado una frase y a continuación visualice las palabras de la frase en columna, seguida cada una del número de letras que la componen. Implementa tres versiones diferentes:

- Sin utilizar ningún string, simplemente procesando la entrada carácter a carácter hasta el final (con `cin.get()`).
- Leyendo la entrada palabra a palabra con el extractor de cadenas.
- Leyendo la frase de una vez con `getline(cin,frase)` y luego procesando el string.

6. Para que los cálculos resultasen más fáciles, los romanos solían utilizar una notación aditiva pura. Un número romano en esta notación es una cadena de dígitos romanos, comenzando por el dígito del valor más alto y terminando con el de valor más pequeño, y su valor se obtenía simplemente como la concatenación de sus dígitos (en esta notación, por ejemplo, el número arábigo 19 se representa como XVIIII, X -10- seguido de VIIII -9-). (Nota: en lugar de usar las expresiones sustractivas, como IX, se usan siempre las aditivas, VIIII.)

Escribe un subprograma `romanoAEntero()` que lea una cadena de caracteres introducida por el teclado y devuelva:

- Un valor que indique si la cadena leída se puede interpretar como un número romano válido en notación aditiva pura, y
- el número arábigo equivalente calculado en caso de ser válido el número romano.

Así, por ejemplo, si se introduce por teclado `MDCCCCLXXXVIIII` el subprograma devolverá un valor que indique que el número romano es válido y el número arábigo 1989. En cambio, si se introduce por teclado `aX`, el subprograma devolverá un valor que indique que no es un número romano válido.

NOTA: La cadena no se podrá interpretar como número romano si contiene algún carácter que no sea un dígito romano. Recuerda que los dígitos romanos y sus valores correspondientes son: I = 1; V = 5; X = 10; L = 50; C = 100; D = 500; M = 1000.

7. Queremos programar el juego de las 3 en raya:

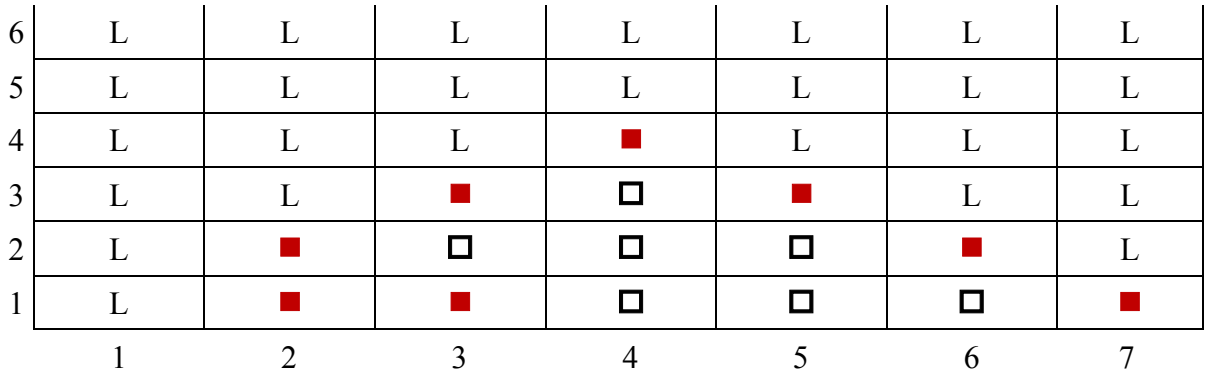
X	O	
	X	X
	O	O

- Define una estructura de datos para representar el tablero de juego.
- Implementa un subprograma para inicializar el tablero con blancos.
- Implementa un subprograma para dibujar el tablero junto con el contenido de cada posición (`'X'`, `'O'`, `' '`).
- Implementa un subprograma que devuelva `true` o `false`, dependiendo de si todo el tablero está ocupado o existen casillas libres.
- Implementa la función Booleana `tresEnRaya(tableroJuego,jugador)` que devuelva `true` en caso de que un jugador tenga la jugada de tres en raya y `false` en caso contrario.

8. Se quiere desarrollar una función que dadas dos matrices cuadradas de enteros, M de 10 x 10 y P de 3 x 3, compruebe si entre todas las submatrices de 3 x 3 que se pueden formar en la matriz M, desplazándose por filas o columnas, existe al menos una que coincida con la matriz P. En ese

caso, se indicarán la fila y la columna de la matriz M en la que empieza la submatriz que coincide. Si hay varias, bastará con indicar la primera.

9. El juego de las 4 en línea consta de un tablero formado por siete columnas y seis filas. En una partida participan dos jugadores, uno con fichas blancas y otro rojas. Inicialmente todas las posiciones del tablero están libres. Cada jugador coloca alternativamente una ficha en una columna. La ficha colocada cae por su propio peso hasta el fondo de la columna correspondiente (primera fila de la columna libre); por ejemplo, en la figura si el jugador Rojo coloca una ficha en la columna 2, la ficha se coloca en la fila 3. La partida la gana el jugador que coloque en primer lugar cuatro de sus fichas en línea horizontal, vertical o en diagonal. La partida queda en tablas si ninguno de los jugadores es capaz de alinear cuatro fichas después de llenar el tablero.



- Codifica las estructuras de datos necesarias para jugar una partida.
- Escribe un subprograma `inicializarJuego()` que quite todas las fichas del tablero y prepare el tablero de juego.
- Escribe un subprograma `colocarFicha()` que dado un jugador (blanco o rojo) y una columna (1 a 7), coloque la ficha en la posición correspondiente.
- Escribe un subprograma `presentarTablero()` que visualice en la pantalla el estado del tablero (como en la figura, excepto la rejilla).
- Escribe un subprograma `comprobarGanador()` que, dado un jugador (blanco o rojo) y una determinada casilla (1 a 6, 1 a 7), determine si hay cuatro fichas del mismo jugador alineadas en horizontal.

10. En un archivo de texto *"inventario.txt"* se guarda información acerca de los productos existentes en un almacén. Para cada producto se guardan los siguientes datos:

- Código del producto (4 dígitos).
- Nombre del producto.
- Número de unidades (entero).
- Precio por unidad (real).

El archivo está ordenado por orden creciente del código del producto.

Se dispone además de otro archivo, *"modificaciones.txt"*, en el que se guardan las modificaciones en el número de artículos que se han producido a lo largo de todo un día. Cada componente del archivo *"modificaciones.txt"* consta de los siguientes campos:

- Código del producto (4 dígitos).
- Código de operación: venta (V), compra (C), o devolución (D)

- Número de unidades (entero).

El archivo *"modificaciones.txt"* está organizado secuencialmente y no se encuentra ordenado de ninguna manera especial. Puede haber varias modificaciones relativas al mismo producto. Es decir, el producto de código 2331 puede tener asociadas, por ejemplo, tres componentes del archivo *"modificaciones.txt"*: la primera puede tratarse de la devolución de 100 unidades, la segunda de la devolución de 50 unidades y la tercera de la venta de 550 unidades.

Se pide:

- Efectuar las declaraciones necesarias para almacenar los datos necesarios.
- Implementar un subprograma que almacene la información del fichero *"inventario.txt"* en las estructuras de datos correspondientes.
- Implementar un subprograma que guarde en fichero la información contenida en la lista del inventario.
- Implementar un subprograma que dado un código de producto devuelva la posición del mismo en la lista del inventario usando la búsqueda binaria.
- Implementar un subprograma que guarde en un nuevo archivo *"inventarioActualizado.txt"* la actualización del archivo "inventario.txt" inicial usando el archivo *"modificaciones.txt"* (las operaciones de venta disminuyen el número de unidades y las de compra y devolución lo aumentan).

11. Se pide construir un programa que muestre por orden de fecha los alquileres de una agencia de alquiler de coches. Se dispone de dos archivos, uno (coches.txt) con la información de códigos (enteros) y nombres (cadenas) de los coches de los que se dispone. La información en el archivo está ordenada por códigos y termina con -1 como código:

```
1325 Seat León
1548 Volkswagen Golf
...
-1
```

El otro archivo (rent.txt) contiene (sin ningún orden) la relación de alquileres que se han contratado (código del coche, fecha en formato AA/MM/DD y días que se ha alquilado). Termina con un -1 como código:

```
7377 13/03/19 1
2176 13/02/11 7
6664 13/03/17 3
3349 13/01/21 1
...
-1
```

El programa deberá empezar cargando la información de cada archivo en sendas listas: una lista de coches (máximo 20) y una lista de alquileres (máximo 100). La lista de coches quedará ordenada por orden de códigos (como en el archivo). Una vez leídas las listas ordenará la lista de alquileres por fechas. Terminará mostrando la información sobre los alquileres (fecha, modelo y días alquilado):

```
13/01/17 Volkswagen Golf 3 día(s)
13/01/21 Opel Zafira 1 día(s)
13/01/31 Opel Vivaro 3 día(s)
```

```
...
13/02/26 Volkswagen Passat 3 día(s)
13/02/27 ERROR: ¡¡¡Modelo inexistente!!!
...
```

El programa deberá hacer uso de los siguientes subprogramas:

- leerModelos(): carga la información del archivo coches.txt en la lista de coches; devuelve true si se ha podido abrir el archivo y false en caso contrario. La lista de coches sólo contendrá la información de este archivo.
- leerAlquileres(): carga la información del archivo rent.txt en la lista de alquileres; devuelve true si se ha podido abrir el archivo y false en caso contrario. La lista de alquileres sólo contendrá la información de este archivo.
- ordenar(): ordena la lista de alquileres por orden de fecha (menor a mayor).
- buscar(): dada la lista de coches y un código, devuelve la posición (índice) del elemento de la lista de coches con ese código; si no se encuentra el código devuelve -1. Debe implementarse como búsqueda binaria.
- mostrar(): dadas ambas listas, muestra la relación de alquileres con el formato mostrado arriba; si no se encuentra un código de coche se notificará el error.